FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Mestrado Integrado em Engenharia Informática e Computação

Métodos Formais em Engenharia de Software

Formal Modeling of a Company with a Distributed Printing Service

Turma 5 Tema 2:
Francisco Lopes ei11056
Tiago Neves up201506203

Date:
05/01/2019

Contents

1. Informal system description and list of requirements

1.1. Informal system description

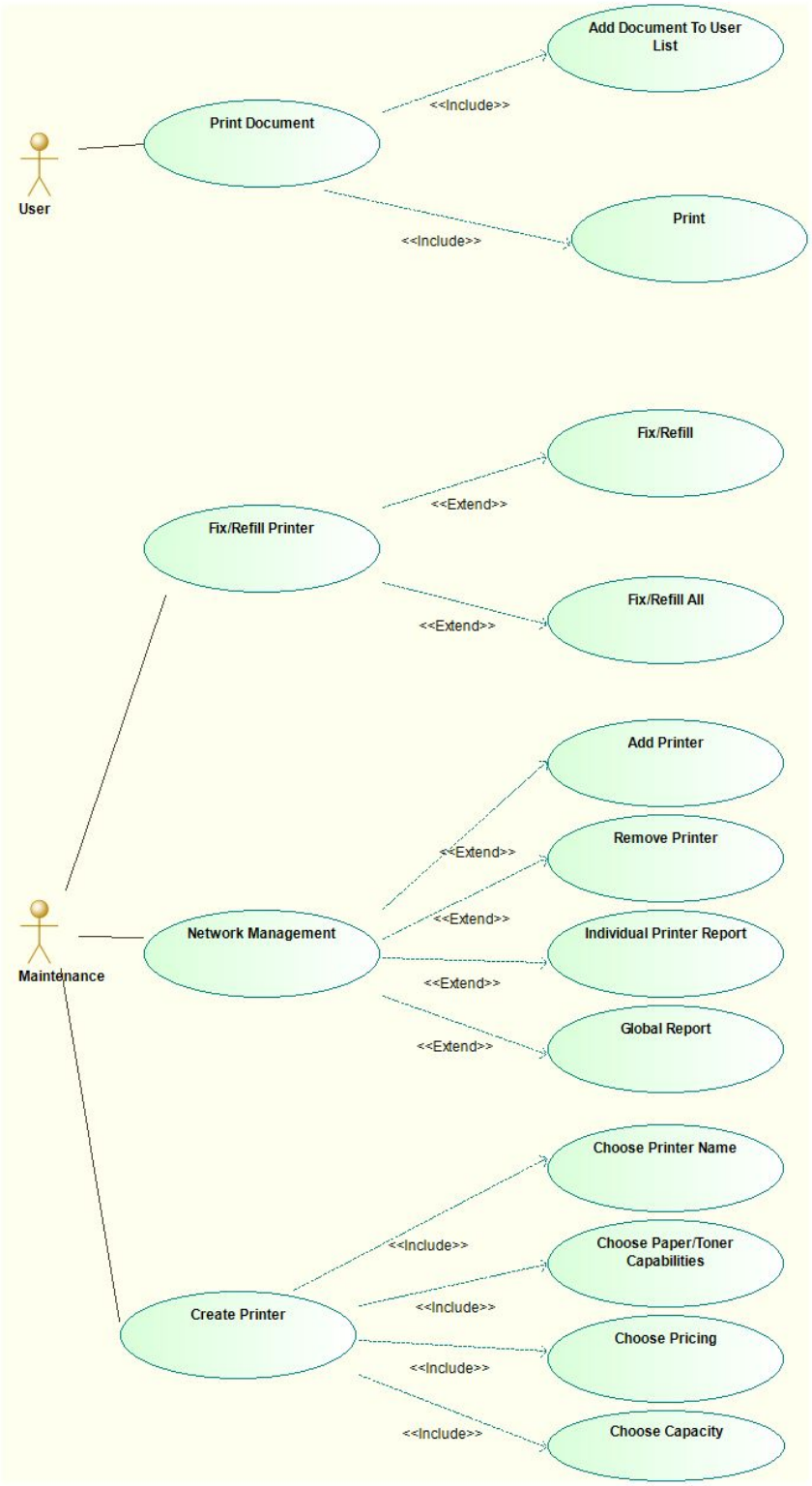
The system to be developed consists of a printer network inside a company capable of printing user documents. Documents can be created with paper/toner requirements (A3 Black, A4 Color, etc) and then added to a user's document list. Afterwards, the user can print documents on their list by choosing a printer to do so. For a successful printing, the printer must be able of printing the document's required paper format and toner and the user must have enough balance to print the maximum amount of pages the printer can print of that document. Finally, there is also a report functionality that outputs metrics about money received and pages printed on each printer of the network.

1.2. List of requirements

ID	**Priority**	**Description**
R1	Mandatory	The User should be able to create documents and add them to their printing list
R2	Mandatory	The User should be able to print documents from their list provided they have enough balance and the printer's capabilities match the document's requirements and paper/toner levels are sufficient for at least one page
R3	Mandatory	The User should have a balance and be able to add additional balance to it
R4	Mandatory	The Maintenance should be able to fix printer malfunction or refill needs
R5	Mandatory	The Maintenance should be able to manage a network of printers by adding/removing printers and be able to request metrics reports of the network

2. Visual UML Model

2.1 Use case model



Picture 1 - Use Case diagram

The major use case scenarios (to be used later as test scenarios) are described next.

Scenario	**Print Document**
Description	Normal scenario for adding a document to the user's printing list and printing it
Pre-conditions	1. Document must be in the user's document list 2. Printer must be capable of using the toner needed by the document 3. Printer must be capable of using the page format needed by the document 4. Printer must have toner left for the document (!= 0) 5. Printer must have paper left for the document (!= 0) 6. Printer must have "operational" status 7. User must have enough balance to print the number of pages the printer can print of the document
Post-conditions	1. Cost is removed from the user's balance *(output)* 2. Printer's resources decrease (paper/toner) *(output)* 3. The pages left to be printed of the document being printed are decreased *(output)* 4. Printer's state must be in accordance with the capacity left of resources *(final system state)*
Steps	1. Create and add a document to the user's printing list 2. Choose a printer in which to print (noting the paper/toner left in the printers) 3. Print the full document or the maximum of the document made possible by the pages or toner left in the printer 4. If the document wasn't fully printed the user can go to another printer and print the remaining pages, or wait for Maintenance to refill the same printer (optional)
Exceptions	(unspecified)

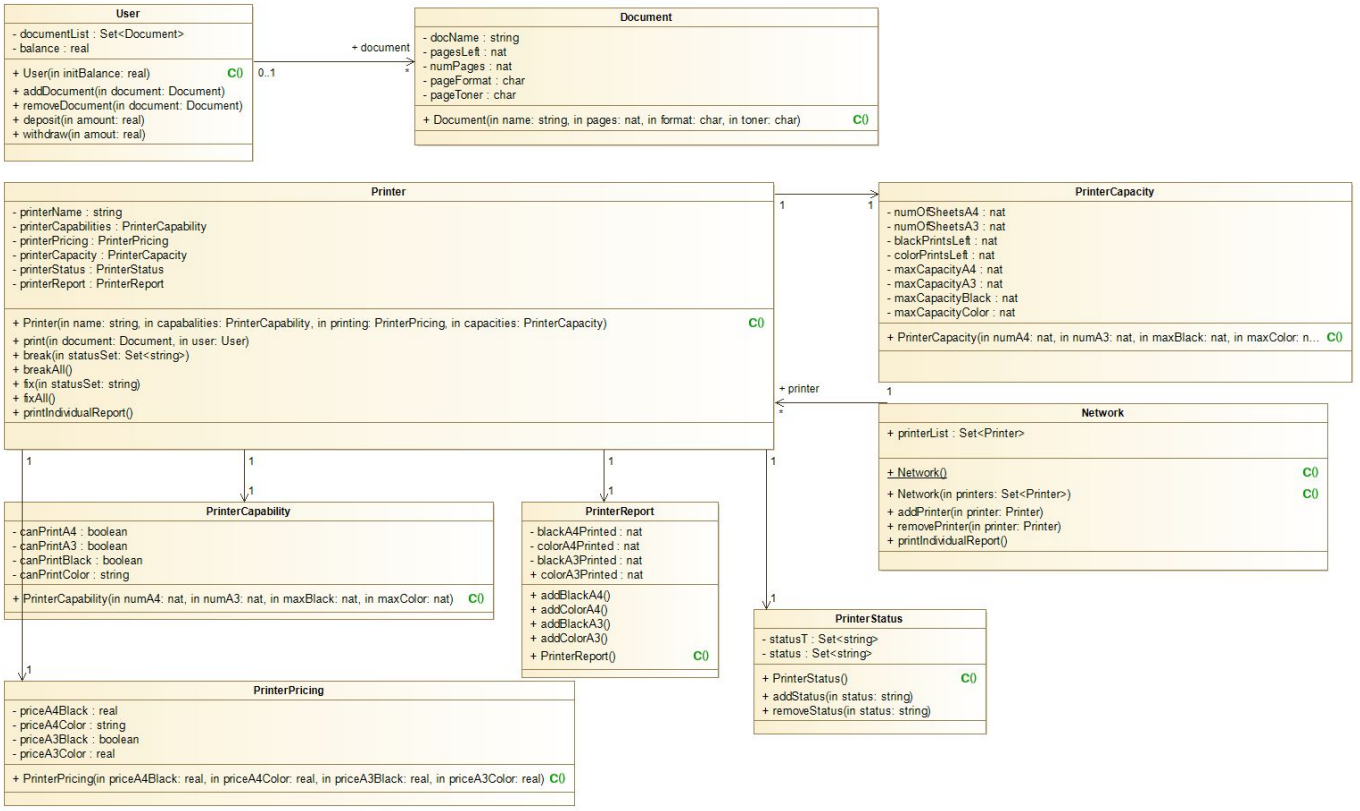
Scenario	**Fix/Refill**
Description	Normal scenario for fixing/refilling a printer
Pre-conditions	1. Printer state must not only be “operational”*(initial system state)*
Post-conditions	1. If the Maintenance chose to fix some state that state was removed (output) 2. If the Maintenance chose to refill some resource it is at the maximum capacity of the printer *(final system state)*
Steps	1. Choose which empty resources of the printer to refill and if the malfunction should be fixed 2. Refill all empty resources that were chosen in the previous step
Exceptions	(unspecified)

Scenario	**Create Printer**
Description	Normal scenario for creating a printer
Pre-conditions	(none)
Post-conditions	1. Printer is operational *(final system state)* 2. Printer’s resources are at the maximum capacity *(final system state)* 3. Printer’s pages printed is at 0 *(final system state)*
Steps	1. Choose a name for the printer 2. Choose the capabilities of the printer (A4/A3, Black/Color) 3. Choose the pricing for each format (A4 Black, A4 Color, A3 Black, A3 Color)
Exceptions	(unspecified)

Scenario	**Network Management**
Description	Normal scenario for managing the network (add printer, remove printer, individual report, global report)
Pre-conditions	1. There exists only one network *(initial system state)*
Post-conditions	(none)
Steps	1. Add a printer to the network *(optional)* 2. Remove a printer from the network *(optional)* 3. Get an individual report from each printer in the network *(optional)* 4. Get a global report from all the printers in the network *(optional)*
Exceptions	(unspecified)

2.2 Class model

The class diagram below omits getters and setters since they would make the diagram harder to interpret.



Picture 2 - Class diagram

Class	Description
Printer	Core model; defines the state variables and operations available to the users
PrinterCapability	Defines printer's capabilities such as paper format and available toners
PrinterCapacity	Defines printer's capacity of each paper format and toner
PrinterPricing	Defines the price per page of each format/toner combination
PrinterReport	Defines printer report that stores metrics about the printer
PrinterStatus	Defines the printer's status such as operational status and malfunctions
Document	Defines Documents that are later associated with Users that can print them at a Printer
Network	Defines a Network of printers that can output metrics of the Network
User	Defines a Userthat can have a list of Documents that can be printed
PrinterTest	Test suite of tests that should pass
PrinterErrorTest	Test suite of tests that fail and validate the preconditions, postconditions and invariants

3. Formal VDM++ model

For proper use of the following code you have to do: New -> Add VDM Library -> IO. We used the IO library for printing the reports and for some of the tests.

3.1. Document

```
class Document
instance variables
  private docName: seq of (char);
  private pagesLeft: nat;
  private numPages: nat;
  private pageFormat: char;
  private pageToner: char;

  inv pageFormat = '3' or pageFormat = '4';
  inv pageToner = 'B' or pageToner = 'C';
  inv numPages > 0;

operations
  -- Constructs a document with specified parameters (docName, numPages,
pageFormat, pageToner)
  -- pageFormat must be '3' or '4'
  -- pageToner must be 'B' or 'C'
  public Document: seq of (char) * nat * char * char ==> Document
              Document(name, pages, format, toner) == (
                docName := name;
                pagesLeft := pages;
                numPages := pages;
                pageFormat := format;
                pageToner := toner;
                return self;
              );

-- GETTERS
  -- Returns the document name
  pure public getDocName: () ==> seq of (char)
       getDocName() == return self.docName;

  -- Returns the number of pages left to print of the document
  pure public getPagesLeft: () ==> nat
       getPagesLeft() == return self.pagesLeft;

  -- Returns the total number of pages of the document
  pure public getNumPages: () ==> nat
       getNumPages() == return self.numPages;

  -- Returns the page format (A3 or A4)
  pure public getPageFormat: () ==> char
```

```
        getPageFormat() == return self.pageFormat;

    -- Returns the page toner (Black or Color)
    pure public getPageToner: () ==> char
        getPageToner() == return self.pageToner;

-- SETTERS
    -- Sets the number of pages left to print of the document
    public setPagesLeft: nat ==> ()
        setPagesLeft(pages) == pagesLeft := pages;

end Document
```

3.2. Network

```
class Network
instance variables
    private printerList: set of (Printer) := {};

operations
    -- Constructs a network using default parameters (empty set of printers)
    public Network: () ==> Network
                Network() == return self;

    -- Constructs a network of printers using the specified printer list
(printerList)
    public Network: set of (Printer) ==> Network
                Network(printers) == (
                    printerList := printers;
                    return self;
                );

    -- Add specified printer to the network, printer name must be unique
    public addPrinter: Printer ==> ()
        addPrinter(printer) == (
        printerList := printerList union {printer};
        )
        pre not exists p2 in set printerList & p2.getPrinterName() =
printer.getPrinterName();

    -- Remove specified printer from the network
    public removePrinter: Printer ==> ()
        removePrinter(printer) == printerList := printerList \ {printer};

    -- Prints the individual report of every printer in the network
    public printIndividualReport: () ==> ()
    printIndividualReport() == (
        for all printer in set printerList do printer.printIndividualReport();
    );
```

```
  -- Prints a global report of the printer network, contains number of pages
printed per format and toner as well as money received from each combination of
those
  public printGlobalReport: () ==> ()
       printGlobalReport() == (

       dcl numA4BlackPrinters: nat := 0;
       dcl numA4ColorPrinters: nat := 0;
       dcl numA3BlackPrinters: nat := 0;
       dcl numA3ColorPrinters: nat := 0;

       dcl sumA4BlackPages: nat := 0;
       dcl sumA4ColorPages: nat := 0;
       dcl sumA3BlackPages: nat := 0;
       dcl sumA3ColorPages: nat := 0;
       dcl sumPages: nat := 0;

       dcl sumA4BlackMoney: real := 0;
       dcl sumA4ColorMoney: real := 0;
       dcl sumA3BlackMoney: real := 0;
       dcl sumA3ColorMoney: real := 0;
       dcl sumMoney: real := 0;

       -- Sum the total pages printed and money received of all network printers
       for all printer in set printerList do (

       -- Parse printer capabilities
       if printer.getPrinterCapabilities().getCanPrintA4() and
printer.getPrinterCapabilities().getCanPrintBlack()
       then numA4BlackPrinters := numA4BlackPrinters + 1;
       if printer.getPrinterCapabilities().getCanPrintA4() and
printer.getPrinterCapabilities().getCanPrintColor()
       then numA4ColorPrinters := numA4ColorPrinters + 1;
       if printer.getPrinterCapabilities().getCanPrintA3() and
printer.getPrinterCapabilities().getCanPrintBlack()
       then numA3BlackPrinters := numA3BlackPrinters + 1;
       if printer.getPrinterCapabilities().getCanPrintA3() and
printer.getPrinterCapabilities().getCanPrintColor()
       then numA3ColorPrinters := numA3ColorPrinters + 1;

       -- Sum pages printed per type
       sumA4BlackPages := sumA4BlackPages +
printer.getPrinterReport().getBlackA4Printed();
       sumA4ColorPages := sumA4ColorPages +
printer.getPrinterReport().getColorA4Printed();
       sumA3BlackPages := sumA3BlackPages +
printer.getPrinterReport().getBlackA3Printed();
       sumA3ColorPages := sumA3ColorPages +
printer.getPrinterReport().getColorA3Printed();
       sumPages := sumPages + printer.sumTotalPagesPrinted();
```

```
        -- Sum money received from each type of print
        sumA4BlackMoney := sumA4BlackMoney +
printer.getPrinterPricing().getPriceA4Black() *
printer.getPrinterReport().getBlackA4Printed();
        sumA4ColorMoney := sumA4ColorMoney +
printer.getPrinterPricing().getPriceA4Color() *
printer.getPrinterReport().getColorA4Printed();
        sumA3BlackMoney := sumA3BlackMoney +
printer.getPrinterPricing().getPriceA3Black() *
printer.getPrinterReport().getBlackA3Printed();
        sumA3ColorMoney := sumA3ColorMoney +
printer.getPrinterPricing().getPriceA3Color() *
printer.getPrinterReport().getColorA3Printed();
        sumMoney := sumMoney + printer.sumTotalMoneyReceived();
        );

        -- Header
        IO`print("Printers in network: ");
        IO`print(card self.printerList);
        IO`print("\n");
        IO`print("A4 Black capable printers: ");
        IO`print(numA4BlackPrinters);
        IO`print("\n");
        IO`print("A4 Color capable printers: ");
        IO`print(numA4ColorPrinters);
        IO`print("\n");
        IO`print("A3 Black capable printers: ");
        IO`print(numA3BlackPrinters);
        IO`print("\n");
        IO`print("A3 Color capable printers: ");
        IO`print(numA3ColorPrinters);
        IO`print("\n");

        -- Pages printed
        IO`print("Number of A4 Black pages printed: ");
        IO`print(sumA4BlackPages);
        IO`print("\n");
        IO`print("Number of A4 Color pages printed: ");
        IO`print(sumA4ColorPages);
        IO`print("\n");
        IO`print("Number of A3 Black pages printed: ");
        IO`print(sumA3BlackPages);
        IO`print("\n");
        IO`print("Number of A3 Color pages printed: ");
        IO`print(sumA3ColorPages);
        IO`print("\n");
        IO`print("Total pages printed: ");
        IO`print(sumPages);
        IO`print("\n");

        -- Money received
```

```
        IO`print("Money from printing A4 Black: ");
        IO`print(sumA4BlackMoney);
        IO`print("\n");
        IO`print("Money from printing A4 Color: ");
        IO`print(sumA4ColorMoney);
        IO`print("\n");
        IO`print("Money from printing A3 Black: ");
        IO`print(sumA3BlackMoney);
        IO`print("\n");
        IO`print("Money from printing A3 Color: ");
        IO`print(sumA3ColorMoney);
        IO`print("\n");
        IO`print("Total money: ");
        IO`print(sumMoney);
        IO`print("\n");
        IO`print("\n");
        );

-- GETTERS
  -- Returns the printer list associated with the network
  public getPrinterList: () ==> set of (Printer)
       getPrinterList() == return self.printerList;

end Network
```

3.3. User

```
class User
instance variables
  private documentList: set of (Document) := {};
  private balance: real;

  inv balance >= 0;

operations
  -- Constructs a user with the specified parameters (balance)
  -- Balance must be greater than 0
  public User: real ==> User
       User(initBal) == (
       balance := initBal;
       return self;
       );

  -- Adds the specified document to the user's document list
  public addDocument: Document ==> ()
       addDocument(doc) == (
          doc.setPagesLeft(doc.getNumPages());
       documentList := documentList union {doc};
       )
       pre forall d in set documentList & doc.getDocName() <> d.getDocName();
```

```
  -- Removes specified document from user's document list
  public removeDocument: Document ==> ()
       removeDocument(doc) == documentList := documentList \ {doc};

  -- Credits the user account with the specified amount
  public deposit: real ==> ()
       deposit(amount) == balance := balance + amount;

  -- Debits the user account with the specified amount
  public withdraw: real ==> ()
       withdraw(amount) == balance := balance - amount;

-- GETTERS
  -- Returns the user's balance
  pure public getBalance: () ==> real
       getBalance() == return self.balance;

  -- Returns the user's document list
  pure public getDocumentList: () ==> set of (Document)
       getDocumentList() == return self.documentList;

end User
```

3.4. Printer

```
class Printer
instance variables

  -- Printer config/status
  private printerName: seq of (char);
  private printerCapabilities: PrinterCapability;
  private printerPricing: PrinterPricing;
  private printerCapacities: PrinterCapacity;
  private printerStatus: PrinterStatus;

  -- Report
  private printerReport: PrinterReport := new PrinterReport();

  -- Printer cannot have capacity of paper/toners it can't use
  inv printerCapabilities.getCanPrintA4() = false =>
printerCapacities.getMaxCapacityA4() = 0;
  inv printerCapabilities.getCanPrintA3() = false =>
printerCapacities.getMaxCapacityA3() = 0;
  inv printerCapabilities.getCanPrintBlack() = false =>
printerCapacities.getMaxCapacityBlack() = 0;
  inv printerCapabilities.getCanPrintColor() = false =>
printerCapacities.getMaxCapacityColor() = 0;

  -- Printer must have capacity of paper/toners it can use
  inv printerCapabilities.getCanPrintA4() = true =>
printerCapacities.getMaxCapacityA4() > 0;
```

```
  inv printerCapabilities.getCanPrintA3() = true =>
printerCapacities.getMaxCapacityA3() > 0;
  inv printerCapabilities.getCanPrintBlack() = true =>
printerCapacities.getMaxCapacityBlack() > 0;
  inv printerCapabilities.getCanPrintColor() = true =>
printerCapacities.getMaxCapacityColor() > 0;

  -- Printer cannot have status types that can't be possible for its capabilities
  inv "needA4" in set printerStatus.getStatus() =>
printerCapabilities.getCanPrintA4() = true;
  inv "needA3" in set printerStatus.getStatus() =>
printerCapabilities.getCanPrintA3() = true;
  inv "needBlackToner" in set printerStatus.getStatus() =>
printerCapabilities.getCanPrintBlack() = true;
  inv "needColorToner" in set printerStatus.getStatus() =>
printerCapabilities.getCanPrintColor() = true;

operations
  -- Constructs a printer object with the specified parameters (printerName,
printerCapabilities, printerPricing, printerCapacities, printerStatus)
  -- printerCapabilities defines the page format and toner the printer has
available
  -- printerPricing defines the price of printing each format using each toner
option
  -- printerCapacities defines the max capacity of each paper format and toner
along with the current capacities
  -- printerStatus defines the state of the printer (broken, operational, needing
refill, etc)
  public Printer: seq of (char) * PrinterCapability * PrinterPricing *
PrinterCapacity * PrinterStatus ==> Printer
       Printer(name, capabilities, pricing, capacities, status) == (
       printerName := name;
       printerCapabilities := capabilities;
       printerPricing := pricing;
       printerCapacities := capacities;
       printerStatus := status;
              return self;
             );

  -- Prints the specified document of the specified user, updates the user's
balance, the printer's toner levels
  -- and the printer's paper levels. Updates the document's pages left with the
printed pages, if all pages
  -- were printed the document is removed from the user's document list. Lastly
updates the printer status
  -- and the printer report with the newly printed page counts.
  -- Constraints:
       -- Document must be in the user's document list
       -- Printer must be capable of using the toner needed by the document
       -- Printer must be capable of using the page format needed by the document
       -- Printer must have toner left for the document (!= 0)
```

```
        -- Printer must have paper left for the document (!= 0)
        -- Printer must have "operational" status
        -- User must have enough balance to print the number of pages the printer
can print of the document (check numPagesToPrint())
    public print: Document * User ==> ()
    print(document, user) == (
        dcl pagesToPrint: nat := numPagesToPrint(document);

        -- Update everything that depends on pagesLeft field
        user.withdraw(calcPrintingCost(document));
        updatePrinterToner(document, pagesToPrint);
      updatePrinterPaper(document, pagesToPrint);

        -- Update document's pages left and remove document from user list if no
more pages left
        document.setPagesLeft(document.getPagesLeft() - pagesToPrint);
        if document.getPagesLeft() = 0
        then user.removeDocument(document);

        -- Update status and report
        updatePrinterStatus();
        updatePrinterReport(document, pagesToPrint);
    )
    pre document in set user.getDocumentList() and
        "operational" in set printerStatus.getStatus() and
        user.getBalance() >= calcPrintingCost(document) and
        checkPrinterTonerCapability(document) and
        checkPrinterPageFormatCapability(document) and
        checkPrinterHasToner(document) and
        checkPrinterHasPaper(document)
    post checkStatusCapacityCoupling();

    -- Computes the total amount of pages that can be printed of the specified
document
    pure public numPagesToPrint: Document ==> nat
        numPagesToPrint(document) == (
        dcl intPages: nat := 0;

        -- Check max pages that the printer can print
        if document.getPageFormat() = '4' and document.getPageToner() = 'B'
        then intPages := minNat(printerCapacities.getNumOfSheetsA4(),
printerCapacities.getBlackPrintsLeft())
        elseif document.getPageFormat() = '4' and document.getPageToner() = 'C'
        then intPages := minNat(printerCapacities.getNumOfSheetsA4(),
printerCapacities.getColorPrintsLeft())
        elseif document.getPageFormat() = '3' and document.getPageToner() = 'B'
        then intPages := minNat(printerCapacities.getNumOfSheetsA3(),
printerCapacities.getBlackPrintsLeft())
        elseif document.getPageFormat() = '3' and document.getPageToner() = 'C'
        then intPages := minNat(printerCapacities.getNumOfSheetsA3(),
printerCapacities.getColorPrintsLeft());
```

```
      -- Return the document's pages left if the printer can print the entire
document or the max the printer can print
      if document.getPagesLeft() > intPages
      then return intPages
      else return document.getPagesLeft();
      );

  -- Returns the minimum number between two natural numbers
  pure private minNat: nat * nat ==> nat
      minNat(a, b) == (
      if a > b
      then return b
      else return a;
      );

  -- Calculates the total amount of money needed to print the maximum possible
pages of the specified document (may not be the entire document)
  pure public calcPrintingCost: Document ==> real
      calcPrintingCost(document) == (
      if document.getPageFormat() = '4' and document.getPageToner() = 'B'
      then return printerPricing.getPriceA4Black() * numPagesToPrint(document)
      elseif document.getPageFormat() = '4' and document.getPageToner() = 'C'
      then return printerPricing.getPriceA4Color() * numPagesToPrint(document)
      elseif document.getPageFormat() = '3' and document.getPageToner() = 'B'
      then return printerPricing.getPriceA3Black() * numPagesToPrint(document)
      elseif document.getPageFormat() = '3' and document.getPageToner() = 'C'
      then return printerPricing.getPriceA3Color() * numPagesToPrint(document)
      else return 0;
      );

  -- Checks printer status/capacity coupling (cannot have a status without the
paper/toner levels being correct for it)
  pure private checkStatusCapacityCoupling: () ==> bool
      checkStatusCapacityCoupling() == (

      if printerCapabilities.getCanPrintA4() then return "needA4" in set
printerStatus.getStatus() <=> printerCapacities.getNumOfSheetsA4() = 0;
      if printerCapabilities.getCanPrintA3() then return "needA3" in set
printerStatus.getStatus() <=> printerCapacities.getNumOfSheetsA3() = 0;
      if printerCapabilities.getCanPrintBlack() then return "needBlackToner" in
set printerStatus.getStatus() <=> printerCapacities.getBlackPrintsLeft() = 0;
      if printerCapabilities.getCanPrintColor() then return "needColorToner" in
set printerStatus.getStatus() <=> printerCapacities.getColorPrintsLeft() = 0;

      return true;
      );

  -- Checks if the printer toner can print the specified document
  -- 'B' means black toner, 'C' means color toner
  pure private checkPrinterTonerCapability: Document ==> bool
```

```
        checkPrinterTonerCapability(document) == (
        if document.getPageToner() = 'B' and printerCapabilities.getCanPrintBlack()
        then return true
        elseif document.getPageToner() = 'C' and
printerCapabilities.getCanPrintColor()
        then return true
        else return false;
        );

    -- Checks if the printer has toner left for the specified document
    -- 'B' means black toner, 'C' means color toner
    pure private checkPrinterHasToner: Document ==> bool
        checkPrinterHasToner(document) == (
        if document.getPageToner() = 'B' and printerCapacities.getBlackPrintsLeft()
= 0
        then return false
        elseif document.getPageToner() = 'C' and
printerCapacities.getColorPrintsLeft() = 0
        then return false
        else return true;
        );

    -- Updates the printer toner used up when printing the specified document
    -- 'B' means black toner, 'C' means color toner
    private updatePrinterToner: Document * nat ==> ()
        updatePrinterToner(document, pagesPrinted) == (
        if document.getPageToner() = 'B'
        then
printerCapacities.setBlackPrintsLeft(printerCapacities.getBlackPrintsLeft() -
pagesPrinted)
        elseif document.getPageToner() = 'C'
        then
printerCapacities.setColorPrintsLeft(printerCapacities.getColorPrintsLeft() -
pagesPrinted);
        );

    -- Checks if the printer can print the page format needed by the specified
document
    -- '4' means A4 and '3' means A3
    pure private checkPrinterPageFormatCapability: Document ==> bool
        checkPrinterPageFormatCapability(document) == (
        if document.getPageFormat() = '4' and printerCapabilities.getCanPrintA4()
        then return true
        elseif document.getPageFormat() = '3' and
printerCapabilities.getCanPrintA3()
        then return true
        else return false;
        );

    -- Checks if the printer has paper left for the specified document
    -- '4' means A4 and '3' means A3
```

```
  pure private checkPrinterHasPaper: Document ==> bool
       checkPrinterHasPaper(document) == (
       if document.getPageFormat() = '4' and printerCapacities.getNumOfSheetsA4()
= 0
       then return false
       elseif document.getPageFormat() = '3' and
printerCapacities.getNumOfSheetsA3() = 0
       then return false
       else return true;
       );

  -- Updates the printer paper used up when printing the specified document
  -- '4' means A4 and '3' means A3
  private updatePrinterPaper: Document * nat ==> ()
       updatePrinterPaper(document, pagesPrinted) == (
       if document.getPageFormat() = '4'
       then
printerCapacities.setNumOfSheetsA4(printerCapacities.getNumOfSheetsA4() -
pagesPrinted)
       elseif document.getPageFormat() = '3'
       then
printerCapacities.setNumOfSheetsA3(printerCapacities.getNumOfSheetsA3() -
pagesPrinted);
       );

  -- Updates the printer status object with refill needs
  private updatePrinterStatus: () ==> ()
       updatePrinterStatus() == (
       if printerCapacities.getNumOfSheetsA4() = 0 and
printerCapabilities.getCanPrintA4()
       then printerStatus.addStatus("needA4");

       if printerCapacities.getNumOfSheetsA3() = 0 and
printerCapabilities.getCanPrintA3()
       then printerStatus.addStatus("needA3");

       if printerCapacities.getBlackPrintsLeft() = 0 and
printerCapabilities.getCanPrintBlack()
       then printerStatus.addStatus("needBlackToner");

       if printerCapacities.getColorPrintsLeft() = 0 and
printerCapabilities.getCanPrintColor()
       then printerStatus.addStatus("needColorToner");
       );

  -- Updates the printer report object with the printed pages
  private updatePrinterReport: Document * nat ==> ()
       updatePrinterReport(document, pages) == (
       if document.getPageFormat() = '4' and document.getPageToner() = 'B'
       then printerReport.addBlackA4(pages)
       elseif document.getPageFormat() = '4' and document.getPageToner() = 'C'
```

```
        then printerReport.addColorA4(pages)
        elseif document.getPageFormat() = '3' and document.getPageToner() = 'B'
        then printerReport.addBlackA3(pages)
        elseif document.getPageFormat() = '3' and document.getPageToner() = 'C'
        then printerReport.addColorA3(pages);
        );

    -- Breaks printer with the specified negative status
    -- Status supplied must be in the available status types (check PrinterStatus
for available status types)
    public break: set of (seq of (char)) ==> ()
        break(statusSet) == (
        for all status in set statusSet do (

        -- Add status related with paper/toner levels
        if status = "needA4"
        then (
                printerCapacities.setNumOfSheetsA4(0);
                printerStatus.addStatus("needA4");
                )
        elseif status = "needA3"
        then (
                printerCapacities.setNumOfSheetsA3(0);
                printerStatus.addStatus("needA3");
                )
        elseif status = "needBlackToner"
        then (
                printerCapacities.setBlackPrintsLeft(0);
                printerStatus.addStatus("needBlackToner");
                )
        elseif status = "needColorToner"
        then (
                printerCapacities.setColorPrintsLeft(0);
                printerStatus.addStatus("needColorToner");
                );

        -- Toggle operational status according to supplied status set
        if status = "needFixing" and "operational" in set printerStatus.getStatus()
        then printerStatus.flipOperationalStatus()
        );
        )
        pre forall s in set statusSet & s in set printerStatus.getStatusT()
        post checkStatusCapacityCoupling();

    -- Sets all negative status types that can be set for the printer's capabilities
    public breakAll: () ==> ()
        breakAll() == (

        if printerCapabilities.getCanPrintA4()
        then (
                printerCapacities.setNumOfSheetsA4(0);
```

```
            printerStatus.addStatus("needA4");
            );

        if printerCapabilities.getCanPrintA3()
        then (
            printerCapacities.setNumOfSheetsA3(0);
            printerStatus.addStatus("needA3");
            );

        if printerCapabilities.getCanPrintBlack()
        then (
            printerCapacities.setBlackPrintsLeft(0);
            printerStatus.addStatus("needBlackToner");
            );

        if printerCapabilities.getCanPrintColor()
        then (
            printerCapacities.setColorPrintsLeft(0);
            printerStatus.addStatus("needColorToner");
            );

        -- Toggle operational status if printer is operational
        if "operational" in set printerStatus.getStatus()
        then printerStatus.flipOperationalStatus()
        )
        post checkStatusCapacityCoupling();

  -- Fixes specified negative status
  -- Status supplied must be in the available status types (check PrinterStatus
for available status types)
  -- Status supplies must be in the current status (can't fix what isn't broken)
  public fix: set of (seq of (char)) ==> ()
      fix(statusSet) == (
      for all status in set statusSet do (

      -- Fix status related to paper/toner levels
      if status = "needA4"
      then (

printerCapacities.setNumOfSheetsA4(printerCapacities.getMaxCapacityA4());
            printerStatus.removeStatus("needA4");
            )
      elseif status = "needA3"
      then (

printerCapacities.setNumOfSheetsA3(printerCapacities.getMaxCapacityA3());
            printerStatus.removeStatus("needA3");
            )
      elseif status = "needBlackToner"
      then (
```

```
printerCapacities.setBlackPrintsLeft(printerCapacities.getMaxCapacityBlack());
              printerStatus.removeStatus("needBlackToner");
              )
       elseif status = "needColorToner"
       then (

printerCapacities.setColorPrintsLeft(printerCapacities.getMaxCapacityColor());
              printerStatus.removeStatus("needColorToner");
              );

       -- Toggle operational status according to supplied status set
       if status = "needFixing" and "needFixing" in set printerStatus.getStatus()
       then printerStatus.flipOperationalStatus()
       );
       )
       pre forall s1 in set statusSet & s1 in set printerStatus.getStatusT() and
       forall s2 in set statusSet & s2 in set printerStatus.getStatus()
       post checkStatusCapacityCoupling();

  -- Sets printer back to "operational" status restoring all paper/toner levels
that can be set for the printer's capabilities
  public fixAll: () ==> ()
       fixAll() == (

       for all status in set printerStatus.getStatus() do (

       if printerCapabilities.getCanPrintA4() and status = "needA4"
       then (

printerCapacities.setNumOfSheetsA4(printerCapacities.getMaxCapacityA4());
              printerStatus.removeStatus("needA4");
              );

       if printerCapabilities.getCanPrintA3() and status = "needA3"
       then (

printerCapacities.setNumOfSheetsA3(printerCapacities.getMaxCapacityA3());
              printerStatus.removeStatus("needA3");
              );

       if printerCapabilities.getCanPrintBlack() and status = "needBlackToner"
       then (

printerCapacities.setBlackPrintsLeft(printerCapacities.getMaxCapacityBlack());
              printerStatus.removeStatus("needBlackToner");
              );

       if printerCapabilities.getCanPrintColor() and status = "needColorToner"
       then (
```

```
printerCapacities.setColorPrintsLeft(printerCapacities.getMaxCapacityColor());
              printerStatus.removeStatus("needColorToner");
              );

       -- Toggle operational status if printer is in need of fixing
       if "needFixing" in set printerStatus.getStatus()
       then printerStatus.flipOperationalStatus()
       )
       )
       post checkStatusCapacityCoupling();

  -- Prints an individual report of the printer, contains number of pages printed
per format and toner as well as money received from each combination of those
  public printIndividualReport: () ==> ()
       printIndividualReport() == (

       -- Header
       IO`print("Name: ");
       IO`print(self.printerName);
       IO`print("\n");

       -- Pages printed
       IO`print("Number of A4 Black pages printed: ");
       if printerCapabilities.getCanPrintA4() and
printerCapabilities.getCanPrintBlack()
       then IO`print(printerReport.getBlackA4Printed())
       else IO`print("N/A");
       IO`print("\n");
       IO`print("Number of A4 Color pages printed: ");
       if printerCapabilities.getCanPrintA4() and
printerCapabilities.getCanPrintColor()
       then IO`print(printerReport.getColorA4Printed())
       else IO`print("N/A");
       IO`print("\n");
       IO`print("Number of A3 Black pages printed: ");
       if printerCapabilities.getCanPrintA3() and
printerCapabilities.getCanPrintBlack()
       then IO`print(printerReport.getBlackA3Printed())
       else IO`print("N/A");
       IO`print("\n");
       IO`print("Number of A3 Color pages printed: ");
       if printerCapabilities.getCanPrintA3() and
printerCapabilities.getCanPrintColor()
       then IO`print(printerReport.getColorA3Printed())
       else IO`print("N/A");
       IO`print("\n");
       IO`print("Total pages printed: ");
       IO`print(sumTotalPagesPrinted());
       IO`print("\n");
```

```
        -- Money received
        IO`print("Money from printing A4 Black: ");
        if printerCapabilities.getCanPrintA4() and
printerCapabilities.getCanPrintBlack()
        then IO`print(printerPricing.getPriceA4Black() *
printerReport.getBlackA4Printed())
        else IO`print("N/A");
        IO`print("\n");
        IO`print("Money from printing A4 Color: ");
        if printerCapabilities.getCanPrintA4() and
printerCapabilities.getCanPrintColor()
        then IO`print(printerPricing.getPriceA4Color() *
printerReport.getColorA4Printed())
        else IO`print("N/A");
        IO`print("\n");
        IO`print("Money from printing A3 Black: ");
        if printerCapabilities.getCanPrintA3() and
printerCapabilities.getCanPrintBlack()
        then IO`print(printerPricing.getPriceA3Black() *
printerReport.getBlackA3Printed())
        else IO`print("N/A");
        IO`print("\n");
        IO`print("Money from printing A3 Color: ");
        if printerCapabilities.getCanPrintA3() and
printerCapabilities.getCanPrintColor()
        then IO`print(printerPricing.getPriceA3Color() *
printerReport.getColorA3Printed())
        else IO`print("N/A");
        IO`print("\n");
        IO`print("Total money: ");
        IO`print(sumTotalMoneyReceived());
        IO`print("\n");
        IO`print("\n");
        );

    -- Returns the sum of the total pages printed at this printer
    public sumTotalPagesPrinted: () ==> nat
        sumTotalPagesPrinted() == return printerReport.getBlackA4Printed() +
                                         printerReport.getColorA4Printed() +
                                  printerReport.getBlackA3Printed() +
                                    printerReport.getColorA3Printed();

    -- Returns the total money received from all printings at this printer
    public sumTotalMoneyReceived: () ==> real
        sumTotalMoneyReceived() == return printerPricing.getPriceA4Black() *
printerReport.getBlackA4Printed() +
                                       printerPricing.getPriceA4Color() *
printerReport.getColorA4Printed() +
                                         printerPricing.getPriceA3Black() *
printerReport.getBlackA3Printed() +
```

```
                                        printerPricing.getPriceA3Color() *
printerReport.getColorA3Printed();

-- GETTERS
  -- Returns the printer name
  pure public getPrinterName: () ==> seq of (char)
       getPrinterName() == return self.printerName;

  -- Returns the printer report object
  public getPrinterReport: () ==> PrinterReport
       getPrinterReport() == return self.printerReport;

  -- Returns the printer pricing object
  public getPrinterPricing: () ==> PrinterPricing
       getPrinterPricing() == return self.printerPricing;

  -- Returns the printer capabilities object
  public getPrinterCapabilities: () ==> PrinterCapability
       getPrinterCapabilities() == return self.printerCapabilities;

  -- Returns the printer capacities object
  public getPrinterCapacities: () ==> PrinterCapacity
       getPrinterCapacities() == return self.printerCapacities;

  -- Returns the printer status object
  public getPrinterStatus: () ==> PrinterStatus
       getPrinterStatus() == return self.printerStatus;

end Printer
```

3.4.1. PrinterCapability

```
class PrinterCapability
instance variables
  private canPrintA4: bool;
  private canPrintA3: bool;
  private canPrintBlack: bool;
  private canPrintColor: bool;

  inv canPrintA4 = true or canPrintA3 = true;
  inv canPrintBlack = true or canPrintColor = true;

operations
  -- Constructs a printer capability object that specifies the page format and
toners available for printing (canPrintA4, canPrintA3, canPrintBlack,
canPrintColor)
  public PrinterCapability: bool * bool * bool * bool ==> PrinterCapability
             PrinterCapability(printA4, printA3, printBlack, printColor) == (
               canPrintA4 := printA4;
               canPrintA3 := printA3;
               canPrintBlack := printBlack;
```

```
            canPrintColor := printColor;
            return self;
        );

-- GETTERS
  -- Returns whether the printer can print in A4
  pure public getCanPrintA4: () ==> bool
       getCanPrintA4() == return self.canPrintA4;

  -- Returns whether the printer can print in A3
  pure public getCanPrintA3: () ==> bool
       getCanPrintA3() == return self.canPrintA3;

  -- Returns whether the printer can print in black
  pure public getCanPrintBlack: () ==> bool
       getCanPrintBlack() == return self.canPrintBlack;

  -- Returns whether the printer can print in color
  pure public getCanPrintColor: () ==> bool
       getCanPrintColor() == return self.canPrintColor;

end PrinterCapability
```

3.4.2. PrinterCapacity

```
class PrinterCapacity
instance variables
  private numOfSheetsA4: nat;
  private numOfSheetsA3: nat;
  private blackPrintsLeft: nat;
  private colorPrintsLeft: nat;
  private maxCapacityA4: nat;
  private maxCapacityA3: nat;
  private maxCapacityBlack: nat;
  private maxCapacityColor: nat;

  inv numOfSheetsA4 <= maxCapacityA4;
  inv numOfSheetsA3 <= maxCapacityA3;
  inv blackPrintsLeft <= maxCapacityBlack;
  inv colorPrintsLeft <= maxCapacityColor;

operations
  -- Constructs a printer capacity object that specifies the max capacity of blank
pages and toner cartridges (maxCapacityA4, maxCapacityA3, maxCapacityBlack,
maxCapacityColor)
  -- Capacity begins at max
  public PrinterCapacity: nat * nat * nat * nat ==> PrinterCapacity
            PrinterCapacity(numA4, numA3, maxBlack, maxColor) == (
              maxCapacityA4 := numA4;
              maxCapacityA3 := numA3;
              maxCapacityBlack := maxBlack;
```

```
            maxCapacityColor := maxColor;
            numOfSheetsA4 := numA4;
            numOfSheetsA3 := numA3;
            blackPrintsLeft := maxBlack;
            colorPrintsLeft := maxColor;
            return self;
        );

-- GETTERS
    -- Returns the max capacity of A4 pages
    pure public getMaxCapacityA4: () ==> nat
        getMaxCapacityA4() == return self.maxCapacityA4;

    -- Returns the max capacity of A3 pages
    pure public getMaxCapacityA3: () ==> nat
        getMaxCapacityA3() == return self.maxCapacityA3;

    -- Returns the max capacity of the black toner
    pure public getMaxCapacityBlack: () ==> nat
        getMaxCapacityBlack() == return self.maxCapacityBlack;

    -- Returns the max capacity of the color toners
    pure public getMaxCapacityColor: () ==> nat
        getMaxCapacityColor() == return self.maxCapacityColor;

    -- Returns the number of A4 pages left in the tray
    pure public getNumOfSheetsA4: () ==> nat
        getNumOfSheetsA4() == return self.numOfSheetsA4;

    -- Returns the number of A3 pages left in the tray
    pure public getNumOfSheetsA3: () ==> nat
        getNumOfSheetsA3() == return self.numOfSheetsA3;

    -- Returns the number of black prints left
    pure public getBlackPrintsLeft: () ==> nat
        getBlackPrintsLeft() == return self.blackPrintsLeft;

    -- Returns the number of color prints left
    pure public getColorPrintsLeft: () ==> nat
        getColorPrintsLeft() == return self.colorPrintsLeft;

-- SETTERS
    -- Set the number of A4 sheets left in the tray
    public setNumOfSheetsA4: nat ==> ()
        setNumOfSheetsA4(numA4) == numOfSheetsA4 := numA4;

    -- Set the number of A3 sheets left in the tray
    public setNumOfSheetsA3: nat ==> ()
        setNumOfSheetsA3(numA3) == numOfSheetsA3 := numA3;

    -- Set the number of black toner left
```

```
  public setBlackPrintsLeft: nat ==> ()
       setBlackPrintsLeft(numBlacks) == blackPrintsLeft := numBlacks;

  -- Set the number of color toner left
  public setColorPrintsLeft: nat ==> ()
       setColorPrintsLeft(numColor) == colorPrintsLeft := numColor;

end PrinterCapacity
```

3.4.3. PrinterPricing

```
class PrinterPricing
instance variables
  private priceA4Black: real;
  private priceA4Color: real;
  private priceA3Black: real;
  private priceA3Color: real;

       inv priceA4Black > 0;
       inv priceA4Color > 0;
       inv priceA3Black > 0;
       inv priceA3Color > 0;

operations
  -- Constructs a printer pricing object that specifies the cost of printing all
available formats (priceA4Black, priceA4Color, priceA3Black, priceA3Color)
  -- All prices must be real numbers greater than 0
  public PrinterPricing: real * real * real * real ==> PrinterPricing
              PrinterPricing(priceA4B, priceA4C, priceA3B, priceA3C) == (
                priceA4Black := priceA4B;
                priceA4Color := priceA4C;
                priceA3Black := priceA3B;
                priceA3Color := priceA3C;
                return self;
              );

-- GETTERS
  -- Returns the page price of printing A4 in black
  pure public getPriceA4Black: () ==> real
       getPriceA4Black() == return self.priceA4Black;

  -- Returns the page price of printing A4 in color
  pure public getPriceA4Color: () ==> real
       getPriceA4Color() == return self.priceA4Color;

  -- Returns the page price of printing A3 in black
  pure public getPriceA3Black: () ==> real
       getPriceA3Black() == return self.priceA3Black;

  -- Returns the page price of printing A3 in color
  pure public getPriceA3Color: () ==> real
```

```
      getPriceA3Color() == return self.priceA3Color;

end PrinterPricing
```

3.4.4. PrinterReport

```
class PrinterReport
instance variables
  private blackA4Printed: nat := 0;
  private colorA4Printed: nat := 0;
  private blackA3Printed: nat := 0;
  private colorA3Printed: nat := 0;

operations
  -- Constructs a printer report with default parameters (0 of all types printed)
  public PrinterReport: () ==> PrinterReport
      PrinterReport() == return self;

  -- Adds the specified number of pages to the total A4 black pages printed
  public addBlackA4: nat ==> ()
      addBlackA4(pages) == blackA4Printed := blackA4Printed + pages;

  -- Adds the specified number of pages to the total A4 color pages printed
  public addColorA4: nat ==> ()
      addColorA4(pages) == colorA4Printed := colorA4Printed + pages;

  -- Adds the specified number of pages to the total A3 black pages printed
  public addBlackA3: nat ==> ()
      addBlackA3(pages) == blackA3Printed := blackA3Printed + pages;

  -- Adds the specified number of pages to the total A3 color pages printed
  public addColorA3: nat ==> ()
      addColorA3(pages) == colorA3Printed := colorA3Printed + pages;

-- GETTERS
  -- Returns the number of A4 black pages printed
  public getBlackA4Printed: () ==> nat
      getBlackA4Printed() == return self.blackA4Printed;

  -- Returns the number of A4 color pages printed
  public getColorA4Printed: () ==> nat
      getColorA4Printed() == return self.colorA4Printed;

  -- Returns the number of A3 black pages printed
  public getBlackA3Printed: () ==> nat
      getBlackA3Printed() == return self.blackA3Printed;

  -- Returns the number of A3 color pages printed
  public getColorA3Printed: () ==> nat
      getColorA3Printed() == return self.colorA3Printed;
```

```
end PrinterReport
```

3.4.5. PrinterStatus

```
class PrinterStatus
instance variables
  private statusT: set of (seq of (char)) := {"needA4", "needA3",
"needBlackToner", "needColorToner", "needFixing", "operational"};
  private status: set of (seq of (char)) := {"operational"};

  -- no broken/fixed printer at the same time
  inv not(exists s1 in set status & s1 = "operational" and
       exists s2 in set status & s2 = "needFixing");
  -- all status must be taken from the available ones
  inv forall s in set status & s in set statusT;
  -- printer must have at least one status
  inv card status > 0;
  -- printer has to have at least "operational" or "needFixing" status
  inv exists s1 in set status & s1 = "operational" or
       exists s2 in set status & s2 = "needFixing";

operations

  -- Constructs a printer status object with default parameters ("operational")
  public PrinterStatus: () ==> PrinterStatus
              PrinterStatus() == return self;

  -- Adds a status to the printer (status must be from the available status)
  public addStatus: seq of (char) ==> ()
       addStatus(stat) == status := status union {stat};

  -- Removes a status from the printer (status must exist in the set)
  public removeStatus: seq of (char) ==> ()
       removeStatus(stat) == (
       status := status \ {stat};
       )
       pre stat in set status;

  -- Flips "operational" to "needFixing" and vice-versa, as to not break the
invariants specified
  public flipOperationalStatus: () ==> ()
       flipOperationalStatus() == (
       dcl newStatus: set of (seq of (char)) := status;
       if "operational" in set status
       then newStatus := newStatus \ {"operational"} union {"needFixing"}
       else newStatus := newStatus \ {"needFixing"} union {"operational"};
       status := newStatus;
       );

-- GETTERS
  -- Returns the printer status set
```

```
  pure public getStatus: () ==> set of (seq of (char))
       getStatus() == return self.status;

  -- Returns the available printer status types
  pure public getStatusT: () ==> set of (seq of (char))
       getStatusT() == return self.statusT;

end PrinterStatus
```

4. Model validation

The next two classes were used for validating the formal mode in VDM++. The “PrinterTest” class covers all important functions and the “PrinterErrorTest” class covers testing the preconditions, postconditions and invariants of the model by supplying invalid parameters or situations.
For proper use of the following code you have to do: New -> Add VDM Library -> IO.
The following table shows which tests fulfill the requirements of the system.

Requirement ID	**Test Name**
R1 - user documents	testUser()
R2 - user can print their own documents	testPrinter()
R3 - user balance	testUser()
R4 - maintenance can fix printer	testPrinter() (break(), fix() section)
R5 - maintenance can manage network	testNetwork()

4.1 PrinterTest

```
class PrinterTest
operations

  -- Checks if a boolean expression is true
  private assertTrue: bool ==> ()
       assertTrue(cond) == return pre cond;

  -- Checks values are equal
  private assertEqual: ? * ? ==> ()
       assertEqual(expected, actual) == return post expected = actual;

  -- Checks values are approximately equal
  private assertApproximatelyEqual: real * real ==> ()
       assertApproximatelyEqual(expected, actual) == (
       dcl delta: real := abs expected - actual;

       if delta < 0.00001
       then assertTrue(true)
       else assertTrue(false);
       );

  -- Testing for PrinterStatus class
  private testPrinterStatus: () ==> ()
       testPrinterStatus() == (
       dcl p1: PrinterStatus := new PrinterStatus();
```

```
        assertEqual(p1.getStatusT(),{"needA4", "needA3", "needBlackToner",
"needColorToner", "needFixing", "operational"});

        p1.addStatus("needA4");
        p1.addStatus("needA3");
        p1.removeStatus("needA4");
        p1.removeStatus("needA3");
        p1.flipOperationalStatus();
        assertEqual(p1.getStatus(), {"needFixing"});

        p1.flipOperationalStatus();
        p1.addStatus("needBlackToner");
        assertEqual(p1.getStatus(),{"operational", "needBlackToner"});
        );

    -- Testing for Document class
    private testDocument: () ==> ()
        testDocument() == (
        dcl d: Document := new Document("testName", 5, '4', 'C');

        assertEqual(d.getDocName(), "testName");
        assertEqual(d.getPageFormat(), '4');
        assertEqual(d.getPageToner(), 'C');
        assertEqual(d.getNumPages(), d.getPagesLeft());
        assertEqual(d.getNumPages(), 5);

        d.setPagesLeft(d.getNumPages() - 2);
        assertEqual(d.getPagesLeft(), 3);
        );

    -- Testing for PrinterCapability class
    private testPrinterCapability: () ==> ()
        testPrinterCapability() == (
        dcl pCapabilityA4Color: PrinterCapability := new PrinterCapability(true,
false, false, true);
        dcl pCapabilityA3Black: PrinterCapability := new PrinterCapability(false,
true, true, false);

        assertEqual(pCapabilityA4Color.getCanPrintA4(), true);
        assertEqual(pCapabilityA4Color.getCanPrintA3(), false);
        assertEqual(pCapabilityA4Color.getCanPrintBlack(), false);
        assertEqual(pCapabilityA4Color.getCanPrintColor(), true);

        assertEqual(pCapabilityA3Black.getCanPrintA4(), false);
        assertEqual(pCapabilityA3Black.getCanPrintA3(), true);
        assertEqual(pCapabilityA3Black.getCanPrintBlack(), true);
        assertEqual(pCapabilityA3Black.getCanPrintColor(), false);
        );

    -- Testing for PrinterCapacity class
    private testPrinterCapacity: () ==> ()
```

```
        testPrinterCapacity() == (
        dcl pCapacity: PrinterCapacity := new PrinterCapacity(7, 8, 9, 10);

        -- Test values are as expected
        assertEqual(pCapacity.getMaxCapacityA4(), 7);
        assertEqual(pCapacity.getMaxCapacityA3(), 8);
        assertEqual(pCapacity.getMaxCapacityBlack(), 9);
       assertEqual(pCapacity.getMaxCapacityColor(), 10);

        assertEqual(pCapacity.getMaxCapacityA4(), pCapacity.getNumOfSheetsA4());
        assertEqual(pCapacity.getMaxCapacityA3(), pCapacity.getNumOfSheetsA3());
       assertEqual(pCapacity.getMaxCapacityBlack(),
pCapacity.getBlackPrintsLeft());
       assertEqual(pCapacity.getMaxCapacityColor(),
pCapacity.getColorPrintsLeft());

        -- Test removing paper/toner
        pCapacity.setNumOfSheetsA4(pCapacity.getNumOfSheetsA4() - 4);
       pCapacity.setNumOfSheetsA3(pCapacity.getNumOfSheetsA3() - 3);
       pCapacity.setBlackPrintsLeft(pCapacity.getBlackPrintsLeft() - 2);
        pCapacity.setColorPrintsLeft(pCapacity.getColorPrintsLeft() - 1);

        assertEqual(pCapacity.getNumOfSheetsA4(), 3);
        assertEqual(pCapacity.getNumOfSheetsA3(), 5);
       assertEqual(pCapacity.getBlackPrintsLeft(), 7);
        assertEqual(pCapacity.getColorPrintsLeft(), 9);
        );

  -- Testing for PrinterPricing class
  private testPrinterPricing: () ==> ()
        testPrinterPricing() == (
        dcl pPricing: PrinterPricing := new PrinterPricing(0.6, 0.7, 0.8, 0.9);

        assertEqual(pPricing.getPriceA4Black(), 0.6);
        assertEqual(pPricing.getPriceA4Color(), 0.7);
        assertEqual(pPricing.getPriceA3Black(), 0.8);
        assertEqual(pPricing.getPriceA3Color(), 0.9);
        );

  -- Testing for PrinterReport class
  private testPrinterReport: () ==> ()
        testPrinterReport() == (
        dcl pReport: PrinterReport := new PrinterReport();

        assertEqual(pReport.getBlackA4Printed(), 0);
        assertEqual(pReport.getColorA4Printed(), 0);
        assertEqual(pReport.getBlackA3Printed(), 0);
        assertEqual(pReport.getColorA3Printed(), 0);

        pReport.addBlackA4(5);
        pReport.addColorA4(6);
```

```
    pReport.addBlackA3(7);
    pReport.addColorA3(8);

    assertEqual(pReport.getBlackA4Printed(), 5);
    assertEqual(pReport.getColorA4Printed(), 6);
    assertEqual(pReport.getBlackA3Printed(), 7);
    assertEqual(pReport.getColorA3Printed(), 8);

    pReport.addBlackA4( 1);
    pReport.addColorA4( 2);
    pReport.addBlackA3( 3);
    pReport.addColorA3( 4);

    assertEqual(pReport.getBlackA4Printed(), 6);
    assertEqual(pReport.getColorA4Printed(), 8);
    assertEqual(pReport.getBlackA3Printed(), 10);
    assertEqual(pReport.getColorA3Printed(), 12);
    );

-- Testing for User class
private testUser: () ==> ()
    testUser() == (
    dcl doc1: Document := new Document("testName1", 10, '3', 'B');
    dcl doc2: Document := new Document("testName2", 20, '4', 'B');
    dcl user: User := new User(0.1);

    -- Test user balance
    assertApproximatelyEqual(user.getBalance(), 0.1);
    user.deposit(1.1);
    assertApproximatelyEqual(user.getBalance(), 1.2);
    user.withdraw(0.2);
    assertApproximatelyEqual(user.getBalance(), 1);

    -- Test user document list
    assertEqual(user.getDocumentList(), {});
    user.addDocument(doc1);
    assertEqual(user.getDocumentList(), {doc1});
    user.addDocument(doc2);
    assertEqual(user.getDocumentList(), {doc1, doc2});
    user.removeDocument(doc1);
    assertEqual(user.getDocumentList(), {doc2});
    user.removeDocument(doc2);
    assertEqual(user.getDocumentList(), {});
    );

-- Testing for Printer class
private testPrinter: () ==> ()
    testPrinter() == (
    dcl user: User := new User(30);

    dcl pPricing4B: PrinterPricing := new PrinterPricing(0.5, 0.8, 0.6, 1);
```

```
        dcl pCapacity4B: PrinterCapacity := new PrinterCapacity(20, 0, 25, 0);
        dcl pCapability4B: PrinterCapability := new PrinterCapability(true, false,
true, false);
        dcl pStatus4B: PrinterStatus := new PrinterStatus();
        dcl printer4B: Printer := new Printer("testPrinterName4B", pCapability4B,
pPricing4B, pCapacity4B, pStatus4B);

        dcl pPricing3C: PrinterPricing := new PrinterPricing(0.5, 0.8, 0.6, 1);
        dcl pCapacity3C: PrinterCapacity := new PrinterCapacity(0, 10, 0, 15);
        dcl pCapability3C: PrinterCapability := new PrinterCapability(false, true,
false, true);
        dcl pStatus3C: PrinterStatus := new PrinterStatus();
        dcl printer3C: Printer := new Printer("testPrinterName3C", pCapability3C,
pPricing3C, pCapacity3C, pStatus3C);

        dcl pPricingAll: PrinterPricing := new PrinterPricing(0.5, 0.8, 0.6, 1);
        dcl pCapacityAll: PrinterCapacity := new PrinterCapacity(20, 20, 25, 25);
        dcl pCapabilityAll: PrinterCapability := new PrinterCapability(true, true,
true, true);
        dcl pStatusAll: PrinterStatus := new PrinterStatus();
        dcl printerAll: Printer := new Printer("testPrinterName4C", pCapabilityAll,
pPricingAll, pCapacityAll, pStatusAll);

        dcl doc1: Document := new Document("doc1", 10, '4', 'B');
        dcl doc2: Document := new Document("doc2", 30, '4', 'B');
        dcl doc3: Document := new Document("doc3", 20, '3', 'C');
        dcl doc4: Document := new Document("doc4", 5, '3', 'C');
        dcl doc5: Document := new Document("doc5", 30, '4', 'C');
        dcl doc6: Document := new Document("doc6", 5, '3', 'B');

        -- Test printer getters
        assertEqual(printer4B.getPrinterName(), "testPrinterName4B");
        assertEqual(printer4B.getPrinterPricing(), pPricing4B);
       assertEqual(printer4B.getPrinterCapabilities(), pCapability4B);
       assertEqual(printer4B.getPrinterCapacities(), pCapacity4B);
        assertEqual(printer4B.getPrinterStatus(), pStatus4B);

        -- Test numPagesToPrint()
       assertEqual(printer4B.numPagesToPrint(doc1), 10);
       assertEqual(printer4B.numPagesToPrint(doc2), 20);
       assertEqual(printer4B.numPagesToPrint(doc3), 0);

       assertEqual(printer3C.numPagesToPrint(doc4), 5);
       assertEqual(printer3C.numPagesToPrint(doc3), 10);

        -- Test calcPrintingCost()
       assertApproximatelyEqual(printer4B.calcPrintingCost(doc1), 10 * 0.5);
        assertApproximatelyEqual(printer3C.calcPrintingCost(doc4), 5 * 1);

       assertApproximatelyEqual(printerAll.calcPrintingCost(doc1), 10 * 0.5);
       assertApproximatelyEqual(printerAll.calcPrintingCost(doc3), 20 * 1);
```

```
        assertApproximatelyEqual(printerAll.calcPrintingCost(doc4), 5 * 1);
       assertApproximatelyEqual(printerAll.calcPrintingCost(doc5), 20 * 0.8);
       assertApproximatelyEqual(printerAll.calcPrintingCost(doc6), 5 * 0.6);

        -- Test break(), fix(), breakAll() and fixAll()
        printer4B.break({"needA4", "needBlackToner", "needFixing"});
       assertEqual(printer4B.getPrinterStatus().getStatus(), {"needFixing",
"needA4", "needBlackToner"});
        printer4B.fix({"needBlackToner", "needA4", "needFixing"});
       assertEqual(printer4B.getPrinterStatus().getStatus(), {"operational"});

        printer3C.break({"needA3", "needColorToner", "needFixing"});
       assertEqual(printer3C.getPrinterStatus().getStatus(), {"needFixing",
"needA3", "needColorToner"});
        printer3C.fix({"needColorToner","needFixing", "needA3"});
       assertEqual(printer3C.getPrinterStatus().getStatus(), {"operational"});

        printerAll.breakAll();
        assertEqual(printerAll.getPrinterStatus().getStatus(), {"needA4", "needA3",
"needBlackToner", "needColorToner", "needFixing"});
        printerAll.fixAll();
        assertEqual(printerAll.getPrinterStatus().getStatus(), {"operational"});

        -- Test print()
        user.addDocument(doc1);
        user.addDocument(doc2);
        user.addDocument(doc3);
        user.addDocument(doc4);
        user.addDocument(doc5);
       user.addDocument(doc6);

        -- Capacity is (20, 20, 25, 25)
        -- Document is 10 A4 Black pages
        -- after printing capacity will be (10, 20, 15, 25)
        -- Document was removed from the user's list and the cost was withdrawn
from the user's balance
        printerAll.print(doc1, user);
       assertEqual(printerAll.sumTotalPagesPrinted(), 10);
       assertEqual(printerAll.sumTotalMoneyReceived(), 10 * 0.5);
       assertEqual(printerAll.getPrinterCapacities().getNumOfSheetsA4(), 10);
       assertEqual(printerAll.getPrinterCapacities().getBlackPrintsLeft(), 15);
       assertEqual(printerAll.getPrinterReport().getBlackA4Printed(), 10);
       assertEqual(printerAll.getPrinterStatus().getStatus(), {"operational"});
        assertEqual(user.getBalance(), 30 - 10 * 0.5);
        assertEqual(user.getDocumentList(), {doc2, doc3, doc4, doc5, doc6});

        -- Capacity is (10, 20, 15, 25)
        -- Document is 20 A3 Color pages
        -- after printing capacity will be (10, 0, 15, 5) so the tray of A3 paper
will be empty
```

```
        -- Document was removed from the user's list and the cost was withdrawn
from the user's balance
        printerAll.print(doc3, user);
       assertEqual(printerAll.sumTotalPagesPrinted(), 10 + 20);
       assertEqual(printerAll.sumTotalMoneyReceived(), 10 * 0.5 + 20 * 1);
       assertEqual(printerAll.getPrinterCapacities().getNumOfSheetsA3(), 0);
       assertEqual(printerAll.getPrinterCapacities().getColorPrintsLeft(), 5);
       assertEqual(printerAll.getPrinterReport().getColorA3Printed(), 20);
        assertEqual(printerAll.getPrinterStatus().getStatus(), {"operational",
"needA3"});
        assertEqual(user.getBalance(), 30 - 10 * 0.5 - 20 * 1);
        assertEqual(user.getDocumentList(), {doc2, doc4, doc5, doc6});

        -- Capacity is (10, 0, 15, 5)
        -- Document is 30 A4 Color pages
        -- after printing capacity will be (5, 0, 15, 0) so the tray of A3 paper
and the color toner will be empty
        -- Document was not removed from the user's list because it wasn't fully
printed and the cost was withdrawn from the user's balance
        printerAll.print(doc5, user);
       assertEqual(printerAll.sumTotalPagesPrinted(), 10 + 20 + 5);
       assertApproximatelyEqual(printerAll.sumTotalMoneyReceived(), 10 * 0.5 + 20 *
1 + 5 * 0.8);
        assertEqual(printerAll.getPrinterCapacities().getNumOfSheetsA4(), 5);
       assertEqual(printerAll.getPrinterCapacities().getColorPrintsLeft(), 0);
       assertEqual(printerAll.getPrinterReport().getColorA4Printed(), 5);
       assertEqual(printerAll.getPrinterStatus().getStatus(), {"operational",
"needA3", "needColorToner"});
        assertEqual(user.getBalance(), 30 - 10 * 0.5 - 20 * 1 - 5 * 0.8);
        assertEqual(user.getDocumentList(), {doc2, doc4, doc5, doc6});

        printerAll.fixAll();
        assertEqual(printerAll.getPrinterCapacities().getNumOfSheetsA4(), 5);
       assertEqual(printerAll.getPrinterCapacities().getNumOfSheetsA3(), 20);
       assertEqual(printerAll.getPrinterCapacities().getBlackPrintsLeft(), 15);
        assertEqual(printerAll.getPrinterCapacities().getColorPrintsLeft(), 25);
        );

  -- Testing for Network class
  private testNetwork: () ==> ()
        testNetwork() == (
        dcl network: Network := new Network();

        dcl pPricing4B: PrinterPricing := new PrinterPricing(0.5, 0.8, 0.6, 1);
        dcl pCapacity4B: PrinterCapacity := new PrinterCapacity(20, 0, 25, 0);
        dcl pCapability4B: PrinterCapability := new PrinterCapability(true, false,
true, false);
        dcl pStatus4B: PrinterStatus := new PrinterStatus();
        dcl printer4B: Printer := new Printer("testPrinterName4B", pCapability4B,
pPricing4B, pCapacity4B, pStatus4B);
```

```
        dcl pPricing3C: PrinterPricing := new PrinterPricing(0.5, 0.8, 0.6, 1);
        dcl pCapacity3C: PrinterCapacity := new PrinterCapacity(0, 10, 0, 15);
        dcl pCapability3C: PrinterCapability := new PrinterCapability(false, true,
false, true);
        dcl pStatus3C: PrinterStatus := new PrinterStatus();
        dcl printer3C: Printer := new Printer("testPrinterName3C", pCapability3C,
pPricing3C, pCapacity3C, pStatus3C);

        dcl pPricingAll: PrinterPricing := new PrinterPricing(0.5, 0.8, 0.6, 1);
        dcl pCapacityAll: PrinterCapacity := new PrinterCapacity(20, 20, 25, 25);
        dcl pCapabilityAll: PrinterCapability := new PrinterCapability(true, true,
true, true);
        dcl pStatusAll: PrinterStatus := new PrinterStatus();
        dcl printerAll: Printer := new Printer("testPrinterName4C", pCapabilityAll,
pPricingAll, pCapacityAll, pStatusAll);

        dcl network2: Network := new Network({printer3C, printerAll});

        -- Test network add/remove printer functionalities
        assertEqual(network.getPrinterList(), {});
        network.addPrinter(printer4B);
        network.addPrinter(printer3C);
        network.addPrinter(printerAll);
        assertEqual(network.getPrinterList(), {printer4B, printer3C, printerAll});
        network.removePrinter(printer4B);
        assertEqual(network.getPrinterList(), {printer3C, printerAll});
        assertEqual(network.getPrinterList(), network2.getPrinterList());
        );

    -- Testing entry point, calls all the test functions in succession
    public static main: () ==> ()
        main() == (
        dcl testSuite: PrinterTest := new PrinterTest();

        testSuite.testDocument();
        testSuite.testPrinterCapability();
        testSuite.testPrinterCapacity();
        testSuite.testPrinterPricing();
        testSuite.testPrinterReport();
        testSuite.testPrinterStatus();
        testSuite.testUser();
        testSuite.testPrinter();
        testSuite.testNetwork();
        );

end PrinterTest
```

4.2 PrinterErrorTest

```
class PrinterErrorTest
operations
```

```
  -- Checks if a boolean expression is true
  private assertTrue: bool ==> ()
       assertTrue(cond) == return pre cond;

  -- Checks values are equal
  private assertEqual: ? * ? ==> ()
       assertEqual(expected, actual) == return post expected = actual;

  -- Checks values are approximately equal
  private assertApproximatelyEqual: real * real ==> ()
       assertApproximatelyEqual(expected, actual) == (
       dcl delta: real := abs expected - actual;

       if delta < 0.00001
       then assertTrue(true)
       else assertTrue(false);
       );

  -- Testing for PrinterStatus class
  -- Fails because "needA5" is not an available status
  private testPrinterStatusUnavailableStatus: () ==> ()
       testPrinterStatusUnavailableStatus() == (
       dcl p1: PrinterStatus := new PrinterStatus();
       p1.addStatus("needA5");
       );

  -- Fails because a printer can't be operational and needing fixing at the same
time
  private testPrinterStatusOperationalAndNeedFixing: () ==> ()
       testPrinterStatusOperationalAndNeedFixing() == (
       dcl p1: PrinterStatus := new PrinterStatus();
         p1.addStatus("needFixing");
       );

  -- Fails because a printer needs to be operational or needing fixing
  private testPrinterStatusIsntOperationalOrNeedFixing: () ==> ()
    testPrinterStatusIsntOperationalOrNeedFixing() == (
       dcl p1: PrinterStatus := new PrinterStatus();
       p1.addStatus("needA4");
       p1.removeStatus("operational");
       );

  -- Testing for Document class
  -- Fails because a document needs a positive (>= 1) number of pages
  private testDocumentPositiveNumPages: () ==> ()
       testDocumentPositiveNumPages() == (
       dcl d: Document := new Document("testName", 0, '4', 'C');
       assertTrue(true);
       assertEqual(d.getDocName(), "testName");
       );
```

```
  -- Fails because a document format isn't '4' or '3' (the available formats)
  private testDocumentUnavailableFormat: () ==> ()
       testDocumentUnavailableFormat() == (
       dcl d: Document := new Document("testName", 1, '5', 'C');
       assertEqual(d.getDocName(),"testName");
       assertTrue(true);
       );

  -- Fails because a document toner isn't 'B' or 'C' (the available toners)
  private testDocumentUnavailableToner: () ==> ()
       testDocumentUnavailableToner() == (
       dcl d: Document := new Document("testName", 1, '4', 'H');
       assertTrue(true);
       assertEqual(d.getDocName(), "testName");
       );

  -- Testing for PrinterCapability class
  -- Fails because a printer doesn't have the capability to print in black or in
color
  private testPrinterCapabilityBlackOrColor: () ==> ()
       testPrinterCapabilityBlackOrColor() == (
       dcl pCapability: PrinterCapability := new PrinterCapability(true, true,
false, false);
       assertTrue(true);
       assertEqual(pCapability.getCanPrintA4(), true);
       );

  -- Fails because a printer doesn't have the capability to print in A3 or in A4
  private testPrinterCapabilityA3OrA4: () ==> ()
       testPrinterCapabilityA3OrA4() == (
       dcl pCapability: PrinterCapability := new PrinterCapability(false, false,
true, false);
       assertTrue(true);
       assertEqual(pCapability.getCanPrintA4(), false);
       );

  -- Testing for PrinterCapacity class
  -- Fails because the number of sheets lefts can't be greater than the max
  private testPrinterCapacityMax: () ==> ()
    testPrinterCapacityMax() == (
       dcl pCapacity: PrinterCapacity := new PrinterCapacity(7, 7, 7, 7);
       pCapacity.setNumOfSheetsA4(8);
       );

  -- Testing for PrinterPricing class
  -- Fails because the price must be greater than zero
  private testPrinterPricingGreaterThanZero: () ==> ()
       testPrinterPricingGreaterThanZero() == (
       dcl pPricing: PrinterPricing := new PrinterPricing(0, 0.7, 0.8, 0.9);
       assertTrue(true);
```

```
        assertEqual(pPricing.getPriceA4Black(), 0);
        );

    -- Testing for User class
    -- Fails because a User balance must always be greater or equal to 0
    private testUserBalanceGreaterOrEqualtoZero: () ==> ()
        testUserBalanceGreaterOrEqualtoZero() == (
        dcl user: User := new User(0.1);
        user.withdraw(0.2);
        );

    -- Testing for Printer class
    -- Fails because the printer has capacity (> 0) on paper/toners it doesn't have
the capability to print
    private testPrinterHasCapacityButNotCapability: () ==> ()
        testPrinterHasCapacityButNotCapability() == (
        dcl pPricing: PrinterPricing := new PrinterPricing(0.5, 0.8, 0.6, 1);
        dcl pCapacity: PrinterCapacity := new PrinterCapacity(20, 1, 25, 0);
        dcl pCapability: PrinterCapability := new PrinterCapability(true, false,
true, false);
        dcl pStatus: PrinterStatus := new PrinterStatus();
        dcl printer: Printer := new Printer("testPrinterName", pCapability,
pPricing, pCapacity, pStatus);
        assertTrue(true);
        assertEqual(printer.getPrinterName(), "testPrinterName");
        );

    -- Fails because the printer doesn't have capacity (> 0) on paper/toners that it
does have the capability to print
    private testPrinterHasCapabilityButNotCapacity: () ==> ()
        testPrinterHasCapabilityButNotCapacity() == (
        dcl pPricing: PrinterPricing := new PrinterPricing(0.5, 0.8, 0.6, 1);
        dcl pCapacity: PrinterCapacity := new PrinterCapacity(0, 0, 25, 0);
        dcl pCapability: PrinterCapability := new PrinterCapability(true, false,
true, false);
        dcl pStatus: PrinterStatus := new PrinterStatus();
        dcl printer: Printer := new Printer("testPrinterName", pCapability,
pPricing, pCapacity, pStatus);
        assertTrue(true);
        assertEqual(printer.getPrinterName(), "testPrinterName");
        );

    -- Fails because the document isn't in the user's list
    private testPrinterUserDoesntHaveDocument: () ==> ()
        testPrinterUserDoesntHaveDocument() == (
        dcl user: User := new User(30);

        dcl pPricing: PrinterPricing := new PrinterPricing(0.5, 0.8, 0.6, 1);
        dcl pCapacity: PrinterCapacity := new PrinterCapacity(20, 0, 25, 0);
        dcl pCapability: PrinterCapability := new PrinterCapability(true, false,
true, false);
```

```
      dcl pStatus: PrinterStatus := new PrinterStatus();
      dcl printer: Printer := new Printer("testPrinterName", pCapability,
pPricing, pCapacity, pStatus);

      dcl doc1: Document := new Document("doc1", 10, '4', 'B');

      printer.print(doc1, user);
      );

  -- Fails because the document format isn't supported by the printer
  private testPrinterFormatNotInCapabilities: () ==> ()
      testPrinterFormatNotInCapabilities() == (
      dcl user: User := new User(30);

      dcl pPricing: PrinterPricing := new PrinterPricing(0.5, 0.8, 0.6, 1);
      dcl pCapacity: PrinterCapacity := new PrinterCapacity(20, 0, 25, 0);
      dcl pCapability: PrinterCapability := new PrinterCapability(true, false,
true, false);
      dcl pStatus: PrinterStatus := new PrinterStatus();
      dcl printer: Printer := new Printer("testPrinterName", pCapability,
pPricing, pCapacity, pStatus);

      dcl doc1: Document := new Document("doc1", 10, '3', 'B');
      user.addDocument(doc1);

      printer.print(doc1, user);
      );

  -- Fails because the document toner isn't supported by the printer
  private testPrinterTonerNotInCapabilities: () ==> ()
      testPrinterTonerNotInCapabilities() == (
      dcl user: User := new User(30);

      dcl pPricing: PrinterPricing := new PrinterPricing(0.5, 0.8, 0.6, 1);
      dcl pCapacity: PrinterCapacity := new PrinterCapacity(20, 0, 25, 0);
      dcl pCapability: PrinterCapability := new PrinterCapability(true, false,
true, false);
      dcl pStatus: PrinterStatus := new PrinterStatus();
      dcl printer: Printer := new Printer("testPrinterName", pCapability,
pPricing, pCapacity, pStatus);

      dcl doc1: Document := new Document("doc1", 10, '4', 'C');
      user.addDocument(doc1);

      printer.print(doc1, user);
      );

  -- Fails because there is no toner left
  private testPrinterNoTonerLeft: () ==> ()
      testPrinterNoTonerLeft() == (
      dcl user: User := new User(30);
```

```
      dcl pPricing: PrinterPricing := new PrinterPricing(0.5, 0.8, 0.6, 1);
      dcl pCapacity: PrinterCapacity := new PrinterCapacity(25, 0, 20, 0);
      dcl pCapability: PrinterCapability := new PrinterCapability(true, false,
true, false);
      dcl pStatus: PrinterStatus := new PrinterStatus();
      dcl printer: Printer := new Printer("testPrinterName", pCapability,
pPricing, pCapacity, pStatus);

      dcl doc1: Document := new Document("doc1", 20, '4', 'B');
      dcl doc2: Document := new Document("doc2", 1, '4', 'B');
      user.addDocument(doc1);
      user.addDocument(doc2);

      printer.print(doc1, user);
      printer.print(doc2, user);
  );

  -- Fails because there is no pages left
  private testPrinterNoPagesLeft: () ==> ()
      testPrinterNoPagesLeft() == (
      dcl user: User := new User(30);

      dcl pPricing: PrinterPricing := new PrinterPricing(0.5, 0.8, 0.6, 1);
      dcl pCapacity: PrinterCapacity := new PrinterCapacity(20, 0, 25, 0);
      dcl pCapability: PrinterCapability := new PrinterCapability(true, false,
true, false);
      dcl pStatus: PrinterStatus := new PrinterStatus();
      dcl printer: Printer := new Printer("testPrinterName", pCapability,
pPricing, pCapacity, pStatus);

      dcl doc1: Document := new Document("doc1", 20, '4', 'B');
      dcl doc2: Document := new Document("doc2", 1, '4', 'B');
      user.addDocument(doc1);
      user.addDocument(doc2);

      printer.print(doc1, user);
      printer.print(doc2, user)
      );

  -- Fails because printer status isn't "operational"
  private testPrinterNotOperational: () ==> ()
      testPrinterNotOperational() == (
      dcl user: User := new User(30);

      dcl pPricing: PrinterPricing := new PrinterPricing(0.5, 0.8, 0.6, 1);
       dcl pCapacity: PrinterCapacity := new PrinterCapacity(20, 0, 25, 0);
      dcl pCapability: PrinterCapability := new PrinterCapability(true, false,
true, false);
      dcl pStatus: PrinterStatus := new PrinterStatus();
```

```
      dcl printer: Printer := new Printer("testPrinterName", pCapability,
pPricing, pCapacity, pStatus);

      dcl doc1: Document := new Document("doc1", 20, '4', 'B');
      user.addDocument(doc1);

      printer.break({"needFixing"});

      printer.print(doc1,user);
      );

  -- Fails because the user doesn't have enough balance
  private testPrinterUserDoesntHaveBalance: () ==> ()
      testPrinterUserDoesntHaveBalance() == (
      dcl user: User := new User(30);

      dcl pPricing: PrinterPricing := new PrinterPricing(0.5, 0.8, 0.6, 1);
      dcl pCapacity: PrinterCapacity := new PrinterCapacity(20, 0, 25, 0);
      dcl pCapability: PrinterCapability := new PrinterCapability(true, false,
true, false);
      dcl pStatus: PrinterStatus := new PrinterStatus();
      dcl printer: Printer := new Printer("testPrinterName", pCapability,
pPricing, pCapacity, pStatus);

      dcl doc1: Document := new Document("doc1", 20, '4', 'B');
      user.addDocument(doc1);

      printer.break({"needFixing"});

      printer.print(doc1, user);
      );

  -- Fails because the printer doesn't have the capability of the status
  private testPrinterStatusInCapabilities: () ==> ()
      testPrinterStatusInCapabilities() == (
      dcl pPricing: PrinterPricing := new PrinterPricing(0.03, 0.14, 0.06, 0.24);
      dcl pCapability: PrinterCapability := new PrinterCapability(true, false,
true, true);
      dcl pCapacity: PrinterCapacity := new PrinterCapacity(45, 0, 50, 5);
      dcl pStatus: PrinterStatus := new PrinterStatus();
      dcl p: Printer := new Printer("p", pCapability, pPricing, pCapacity,
pStatus);

      IO`println(p.getPrinterStatus().getStatus());
      p.break({"needFixing", "needA3"});
      );

  -- Testing entry point, all of these tests should fail, test one at a time
  public static main: () ==> ()
      main() == (
      dcl testSuite: PrinterErrorTest := new PrinterErrorTest();
```

```
        -- PrinterStatus
        --testSuite.testPrinterStatusUnavailableStatus();
        --testSuite.testPrinterStatusOperationalAndNeedFixing();
        --testSuite.testPrinterStatusIsntOperationalOrNeedFixing();

        -- Document
        --testSuite.testDocumentPositiveNumPages();
        --testSuite.testDocumentUnavailableFormat();
        --testSuite.testDocumentUnavailableToner();

        -- PrinterCapability
        --testSuite.testPrinterCapabilityBlackOrColor();
        --testSuite.testPrinterCapabilityA3OrA4();

        -- PrinterCapacity
       --testSuite.testPrinterCapacityMax();

        -- PrinterPricing
        --testSuite.testPrinterPricingGreaterThanZero();

        -- User
        --testSuite.testUserBalanceGreaterOrEqualtoZero();

        -- Printer
        --testSuite.testPrinterHasCapacityButNotCapability();
        --testSuite.testPrinterHasCapabilityButNotCapacity();
        --testSuite.testPrinterUserDoesntHaveDocument();
        --testSuite.testPrinterFormatNotInCapabilities();
        --testSuite.testPrinterTonerNotInCapabilities();
        --testSuite.testPrinterNoTonerLeft();
        --testSuite.testPrinterNoPagesLeft();
        --testSuite.testPrinterNotOperational();
       testSuite.testPrinterStatusInCapabilities();
        );

end PrinterErrorTest
```

5. Model verification

5.1. Example of domain verification

One of the proof obligations generated by Overture is:

No.	PO Name	Type
46	User`addDocument(Document)	state invariant holds

The code under analysis is:

```
  -- Adds the specified document to the user's document list
  public addDocument: Document ==> ()
       addDocument(doc) == (
          doc.setPagesLeft(doc.getNumPages());
       documentList := documentList union {doc};
       )
       pre forall d in set documentList & doc.getDocName() <> d.getDocName();
```

For the precondition to hold we must ensure that the document name of the document being added must be different from all other document names in the user's document list.

```
(forall doc:Document & ((forall d in set documentList & ((doc.getDocName)() <>
(d.getDocName)())))
```

5.2. Example of invariant verification

Another proof obligations generated by Overture is:

No.	PO Name	Type
49	User`withdraw(real)	state invariant holds

The code under analysis is:

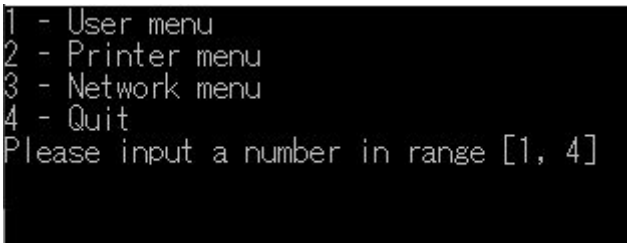
```
  -- Debits the user account with the specified amount
  public withdraw: real ==> ()
       withdraw(amount) == balance := balance - amount;
```

The invariant that must hold is "`inv balance >= 0;`"

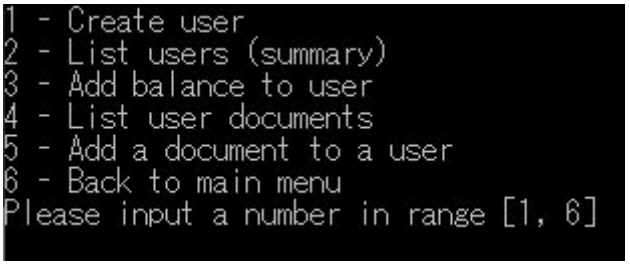
For the invariant to hold we must ensure that "balance - amount >= 0", that is "balance >= amount".

6. Code generation

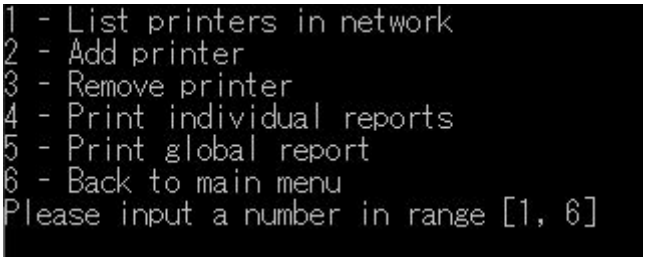
The Java code was generated by using the code conversion tool provided by Overture. The generated code was imported into a new Eclipse Java project and the interface code was developed on top of the VDM++ converted code, which means that the VDM++ code is completely plug and play, there is no need to change the VDM++ generated code. This makes it so formal model updates are possible by simply replacing the files in the Java project with the newly generated VDM++ code.
In the interest of readability the “IO.print” function used in the VDM++ code was replaced with “System.out.print” calls, the former printed the String quotes as well as the text which made the output less readable.
The finalized interface allows testing of every main functionality of the VDM++ formal model, it allows User handling, Document addition, Printer handling, printing documents, Network reports, etc.



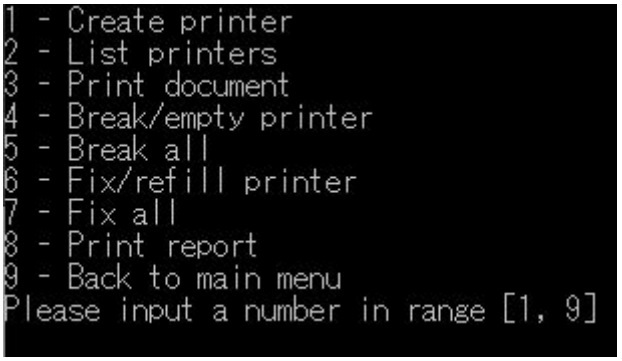
Picture 3 - Main Menu of the interface



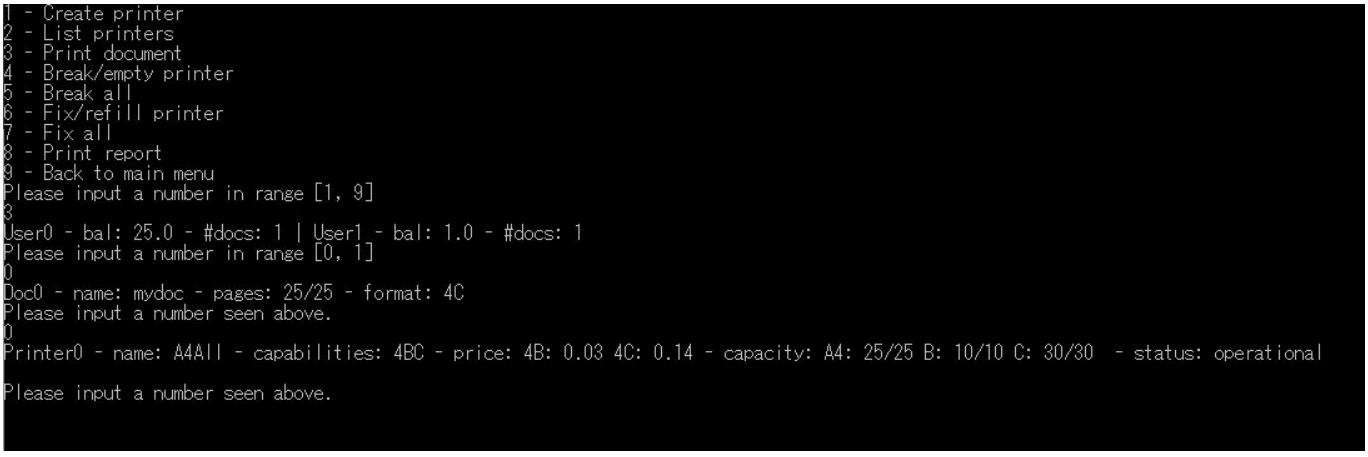
Picture 4 - User Menu of the interface



Picture 5 - Network Menu of the interface



Picture 6 - Printer Menu of the interface



Picture 7 - Print() usage

7. Conclusions

The formal model that was developed covers all requirements and restricts most invalid interactions by means of preconditions, postconditions or invariants, a user is hard-pressed to find invalid interactions with the VDM++ model. The Java interface developed makes it impossible to misuse the VDM++ converted code by only allowing the user to perform valid interactions with the model.
The testing approach began by achieving coverage on all main functionalities (requirements too) and then creating a separate file for tests that were meant to fail the preconditions, postconditions and invariants specified. In the end, pretty much the entirety of the model was tested in expected scenarios and unexpected scenarios.

For future work, additional print() complexity could be implemented by allowing the user to specify printer properties much like in real life such as both sides printing and better paper level handling since the printer currently assumes one page uses one sheet of paper when it could use two pages per paper sheet.

Division of effort:

Francisco:	VDM++ / VDM++ Error tests sketch / Java	(50%)
Tiago:	VDM++ / VDM++ Tests / VDM++ Error tests implementation	(50%)

Overall development time: ~25h

8. References

1. VDM-10 Language Manual, Peter Gorm Larsen et al, Overture Technical Report Series No. TR-001, March 2014
2. Overture VDM-10 Tool Support: User Guide, Peter Gorm Larsen et al, Overture Technical Report Series No. TR-002, February 2018
3. Overture tool web site, http://overturetool.org